DRAFT 1, November 10, 2000

A Self-Organizing Neural Network for Job Scheduling in Distributed Systems

Harvey B. Newman , Iosif C. Legrand

Charles C. Lauristsen Laboratory of High Energy Physics
California Institute of Technology
Pasadena, CA 91125, USA

1. Introduction

Efficient job scheduling policies in large distributed systems, which evolve dynamically, is a challenging task. It requires to analyze a large number of parameters describing the jobs and the time dependent state of the system. The problem is even more difficult when not all these parameters are correctly identified, or when the knowledge about the state of a large distributed system is incomplete or/and known with a certain delay in the past.

The aim of this study is to describe a possible approach for the job-scheduling task in distributed architectures, as a system able to dynamically learn and cluster information in a large dimensional parameter space. This dynamic scheduling system should be seen as an adaptive middle layer software, aware of current available resources and based on the "past experience" to optimize the job performance and resource utilization. We consider such an approach for distributing jobs between regional centers.

The MONARC Simulation tool [1] was extended allowing to dynamically load job scheduling modules for each Regional Center in the simulation frame. This offers the possibility to study practically any job-scheduling scheme and evaluate its performance.

A self-organizing neural network scheduling system may offer a possible solution for an effective use of resources for the off-line data processing jobs for future HEP experiments. These data processing jobs need random access to very large amounts of data, which are assumed to be organized and managed by distributed federations of OODB systems. Such a scheduling system may also take into account the way data are distributed among regional centers and to provide useful information for data replication polices.

2. A "classical" model

A large number of parameters, most of them time dependent, must be used for the job scheduling problem. The following notations will be used to classify them in a few major classes:

{J}	Vector of parameters describing the estimated values for a data processing job.
{S(t)}	Vector of parameters describing the state of the local Regional Center (processing farm) at a certain moment in time.
{R(t)}	Vector of parameters describing all the "external" / connected Regional Centers .
{D}	Set of parameters describing the scheduling decision for each job
{X}	Set of parameters used to quantify the performance obtained after each job was executed.

A “classical” scheme to perform job scheduling is based on a set of rules using (part of) the parameters and a list of empirical constraints based on experience. It can be implemented as a long set of hard coded comparisons to achieve a scheduling decision for each job. In general, it can be represented as a function, which may depend on large numbers of parameters describing the state of the systems and the jobs.

$$\mathbf{D = F\,(J\,,S\,,R\,)}$$

After a job is executed based on this decision, a performance evaluation can be done to quantify it, and to fill the components of **X.**
A schematic view of such a decision making scheme is shown in Figure 1.

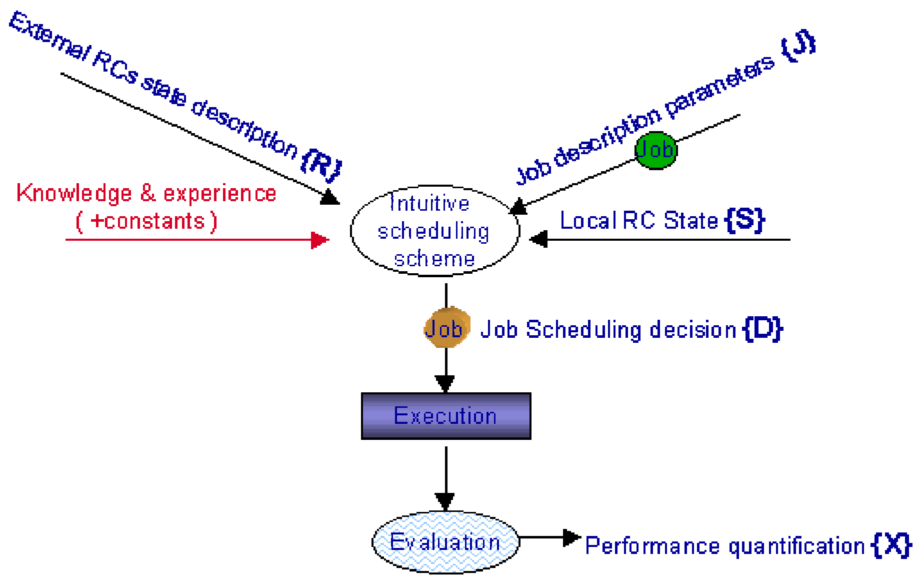
External RCs state description {R}
Job description parameters {J}
Job
Knowledge & experience
(+constants)
Intuitive scheduling scheme
Local RC State {S}
Job
Job Scheduling decision {D}
Execution
Evaluation
Performance quantification {X}

Figure 1. Schematic view of a scheduling decision scheme using an “intuitive” knowledge

3. A self organizing model

This approach is based on using the “past experience” from jobs that have been executed to create a dynamic decision making scheme. A competitive learning algorithm is used to “cluster” correlated information in the multi-dimensional input space defined by the parameters describing the systems, the jobs, the decisions and the results (J,S,R,D and X).
In the area of competitive learning a quite large number of modes exist which may have similar goals but differ considerably in the way they work or the implementation is done to solve certain problems. In our case, a feature mapping architecture able to identify correlation and cluster data distributed in a high-dimensional input space is done using, a growing self organizing network [2,3]. The incremental network models do not have a predefined structure (as in the case of self-organizing maps) and the addition and removal of neurons from the structure are done as part of the learning procedure. Known also as a neural gas, due to the fact that it does not a have a static topological structure, such structure can offer an effective approach for feature extraction in multi dimensional space. The aim of the learning process is to cluster the input data into a set of partitions such that the inta-cluster variance remains small compared with the inter-cluster variance and to estimate the probability density function. This clustering scheme seems possible as we expert a strong correlation between the parameters involved. The strategies in modifying the network topology during the learning process (adding, deleting and clustering

neurons) are to improve the pattern identification in the data and at the same time to try to increase the entire entropy. This will ensure that the network correctly describes the parts of the parameter space, which are populated with low probability. The processes of clustering neurons during the learning may be compared with creating molecules in a gas. Close neurons are connected, and the update procedure for one neuron will also influence the neighbors. A neuron connection also has an age, and if the two neurons connected are not the closest for a large number of data samples, such a connection will be removed. Creating new connections and deleting the "old" ones, makes the topology of this self-organizing network able to better learn complex data patterns than a self-organizing map.
Compared with a "classical" model we expect that such an approach may offer a better way to analyze the possible options and can evolve and improve itself dynamically. This competitive learning algorithm should provide an efficient clustering scheme and learn the correlation between data.

4. A very simple toy example

This over simplified example aims to schematically present the way a self-organizing network can identify correlations and may be used to cluster the information.
We assume that the time to execute a job in the local farm having a certain load (□) is:

$$t_l = t_0(1 + f(\alpha))$$

Where t_0 is the theoretical time to perform the job and f(□) describes the effect of the farm load in the job execution time. If the job is executed on a remote site, an extra factor (□□□□□) is introduced reducing the response time:

$$t_r = t_0\beta(1 + f(\alpha_r))$$

The ratio between the assumed execution time and the theoretical one (t_0) for a logarithmic load function is presented in Figure 2.

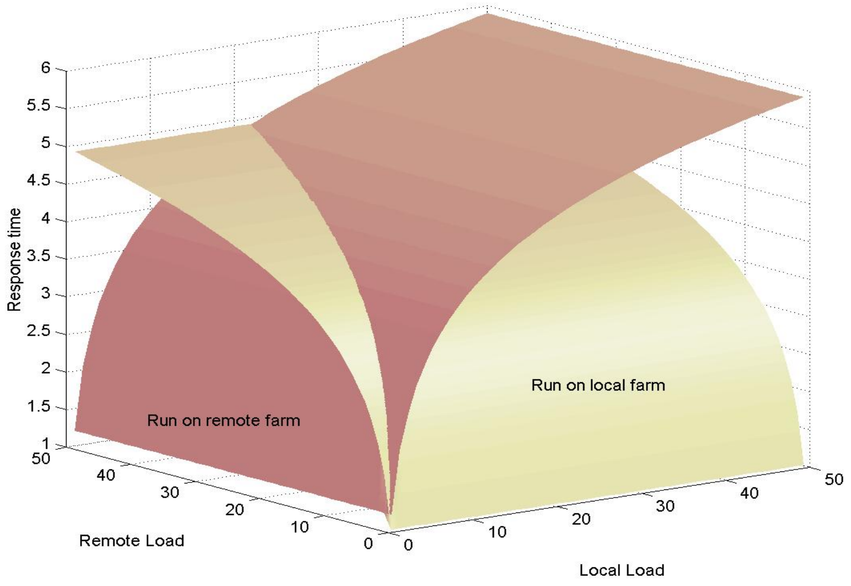

Figure 2. A simple example of the execution time for jobs executed on local or remote systems having different load factors.

The way a competitive learning algorithm for a neural gas like system performs for this problem is shown in Figure 3. “Neurones” connected by lines are relatively close in the parameter space.

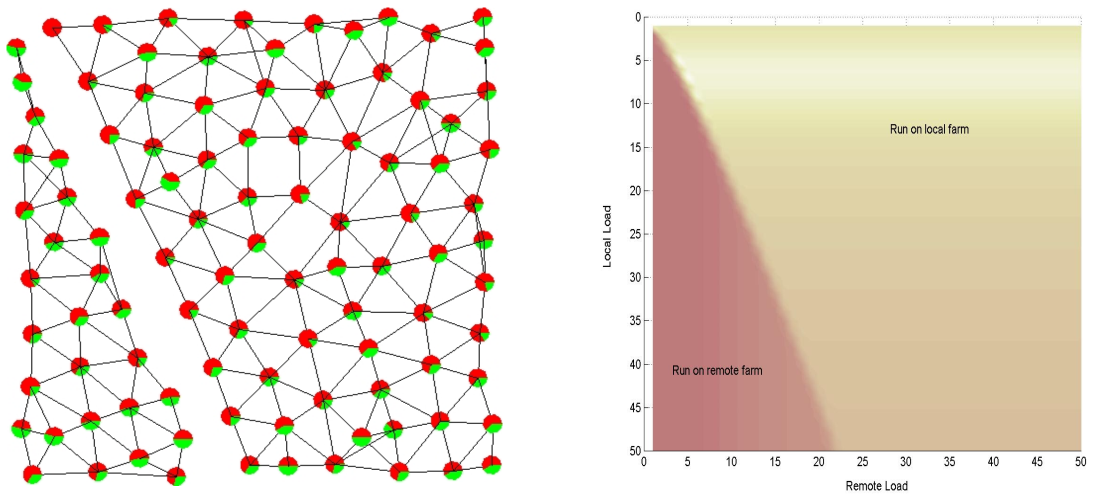

Figure 3. The self-organizing clusters compared with the projected decision types to obtain the best response time.

The two “clusters” corresponding to the two options of where to submit the jobs are well separated and similar with the theoretical separation, and the topology created by the self-organizing network has a very similar mapping.

5. The Scheduling Decision

The decision for future jobs should be based on identifying the clusters in the total parameter space, which are close to the hyper plane defined in this space by the **{J} {S}** subset (parameters known before the job is submitted). In this way the decision can be done evaluating the typical performances of this list of close clusters and chose a decision set which meets the expected / available performance / resources and cost.
However, the self-organizing network has at the beginning a quite limited “knowledge” in the parameter space and exploring efficiently other regions is a quite difficult task.

In this approach for scheduling, the difficult part is not the learning from previous experience, but making decisions when the system does not know (never tried) all possible options for a certain state of the system [2]. Even more difficult is to bring the system into a certain load state, which may require a long sequence of successive decisions taken to achieve it (like in a strategic game problem). The result of each decision is seen only after the job is finished, which also adds to the complexity of quantifying the effect of this decision. For this reason a relatively short history of the previous decisions taken by the system are used also as input parameters in the learning process. We assume a relative long sequence of decision (a policy) can be described with a set of a few points decision history, which partially overlap. This means that we can build a trajectory in the decision space by small segments that partially overlap.

Quite often (at least at the beginning) the decision has to be done without having estimation from the Self-Organizing network for all possible decision options. In this case the strategy used in these examples was to use the best option from previous experience if it seems to provide a better cost function that the mean value from “previous day”. In the case when no information exists about all the possible decision options, and the expected value of the cost function is smaller than the last mean value, a random decision between the remaining options is taken. It is also possible (mainly at the beginning of learning) that the network does not have any cluster near the current state of the system and in this case a random decision is taken.

6. Evaluating the self organizing scheduling with the MONARC Simulation Tool

The diagram in Figure 4 shows the main parts for testing and evaluating the self-organizing scheduling scheme using the MONARC simulation tool. A simple scheduling algorithm is used to initialize the system, and after that the learning procedure combined with the decision scheme described above are stated. After each new job's execution is simulated, the set of **{X}** parameters are evaluated, and together with all the input and decision parameters are kept and used for the future learning procedure. All the past experience is used for further learning as a continuum, independent process.

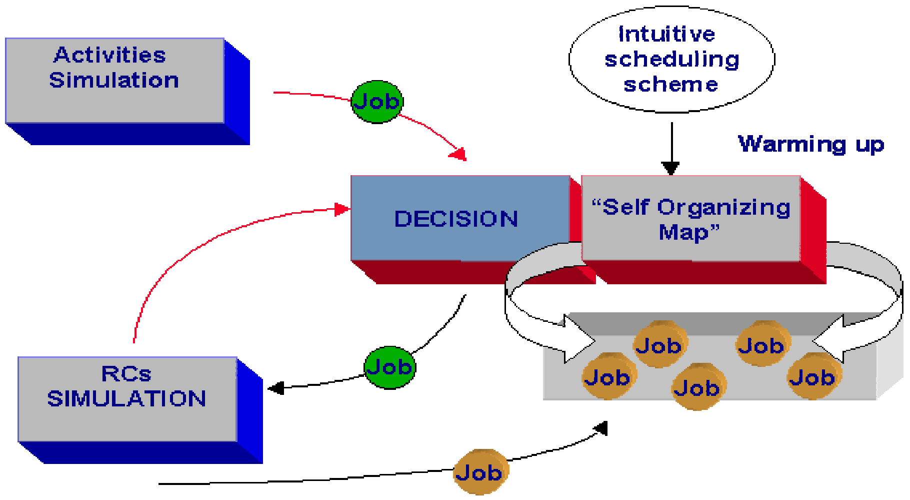

Figure 4. A schematic representation of the main parts used to evaluate a self-organizing scheduling system with the MONARC simulation tool.

The self-organizing scheduling scheme should improve the job execution parameters and resource utilization in time and also to dynamically adapt its self to changes in the system configuration. It may require a "classical" scheduling algorithm as a starting point with the aim to dynamically improve it in time.

7. An Example

We consider three Regional Centers ("caltech", "kek", "cern") connected as in Figure 5. For two of them ("caltech" and "kek") during a certain period of time, every day, the number of data processing jobs submitted for execution exceed the locally available processing power while the third one ("cern") does not have any activity.

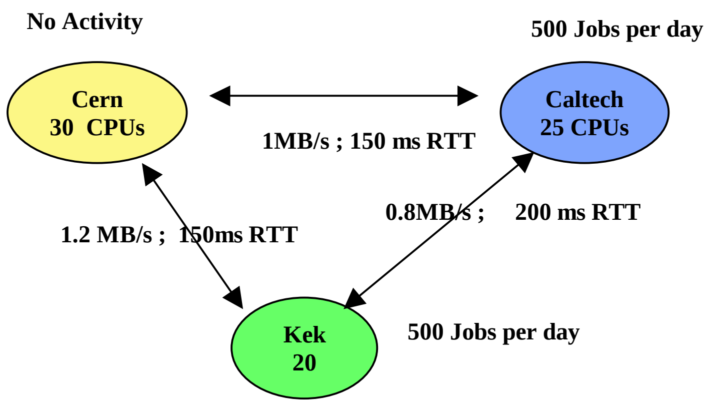

Figure 5. Schematic view of the configuration used to evaluate a self-organizing scheduling.

At Day=0 (initial condition) a “classical” scheduling algorithm is used and all the jobs are submitted only to the local regional center. As the rate of sending jobs into the local regional center during a period of time exceed the local available processing power, many jobs are put into an execution queue. This job queuing makes the mean value for the turnaround time to be quite modest. In Figure 6 is shown the evolution for one day of the number of jobs in each Regional Center as well as the distribution of the turnaround time when all the jobs are executed locally.

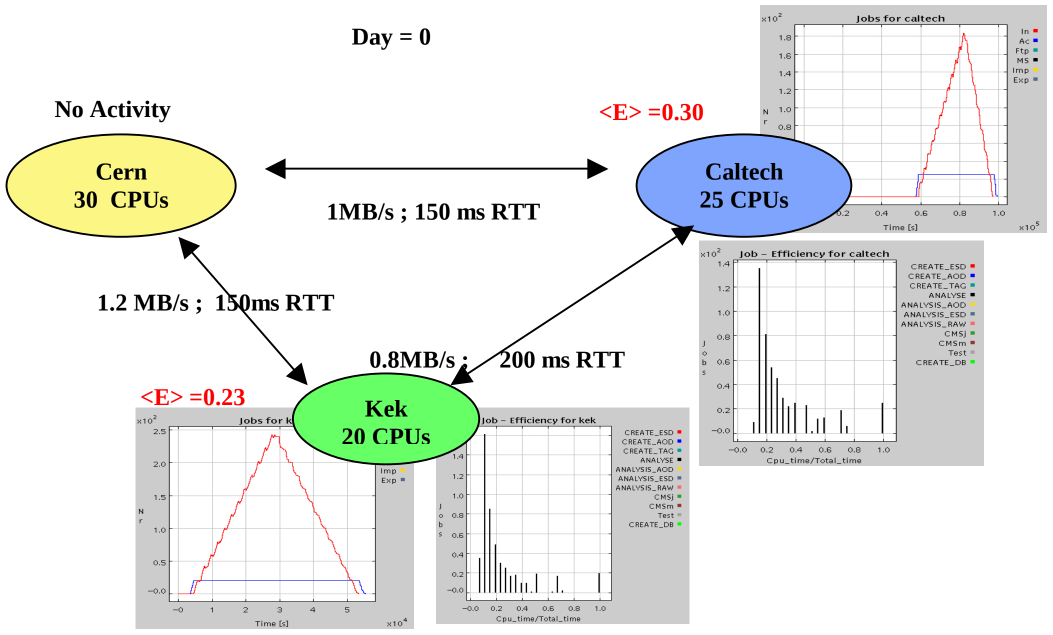

Figure 6. The job evolution (for one day) and the distribution of the turnaround time for the case when all the jobs are executed locally. It can be seen that the number of jobs in the waiting queue (red line) is significant and for this reason the mean value for turnaround time is modest. The blue line describes the number of running jobs.

In the case a job is exported to an other regional center, we considered that the job works with data from its original Regional Center. Due to bandwidth limitation and much higher RTT the execution time on a remote site is longer than running it locally when the processing power is not overloaded. Therefore there is a penalty when jobs are executed on a remote site.

The decision making scheme together with the self organizing networks are “learning” what happens if jobs are exported, and try to provide a better mean return time for all jobs in each Regional Center. In other words this problem can be seen as finding the right amount of jobs which can still be kept in the queue and when it is better to export them. However, due to bandwidth limitation and response time from the data a base server this problem is more complex than simply finding the optimum length of a queuing system. In this case the parameters used in each center for the learning process were: the number of events per job (we used a normal distribution with 10% SD), the CPU load, its derivative, and the length of the waiting queue in the local center and all the remote sites. A set of four-point histories for all of these parameters was used to create the parameter space for the decision. The reason for adding previous decisions in this space is to construct “policy trajectories” as small overlapping segments.

After several days in which the usage pattern was similar as in the first day, the self-organizing scheduler has investigated new possible options and tried to optimize the turnaround time for jobs. The procedure to evaluate new options (exporting jobs) and the way it evolved during the training is presented in Figure 7. It was done using the following scheme: 1) if no nearby cluster in parameter space was found for a job, it was submitted randomly to a regional center; 2) if the number of nearby clusters was less that all possible options and the best return time was better than the mean turnaround time from the previous day, this center was selected (Partially NN) ; 3) if the number of near by clusters was less that all possible options and the best return time was not better than the mean turnaround time from the previous day, a random center from the remaining option was selected (Inverse NN); 4) if the number of nearby clusters was equal to the number of possible options, the center with "known" best turnaround time was selected (Full NN).

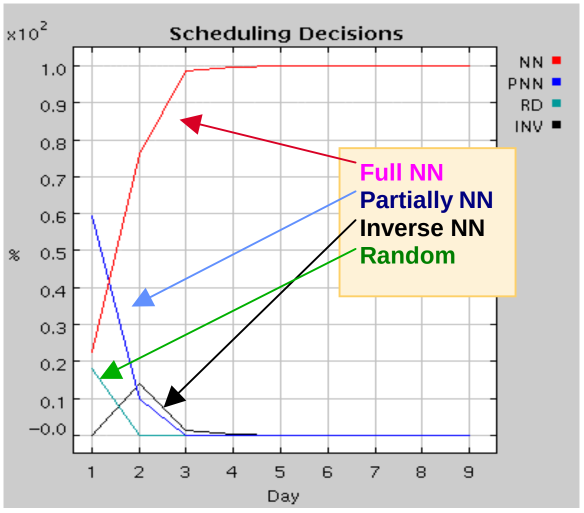

Figure 7. The evolution during the learning of the types of decision done by the scheduling system.

The way jobs were submitted and the distribution for the turnaround time after several days of training is schematically shown in figure 8.

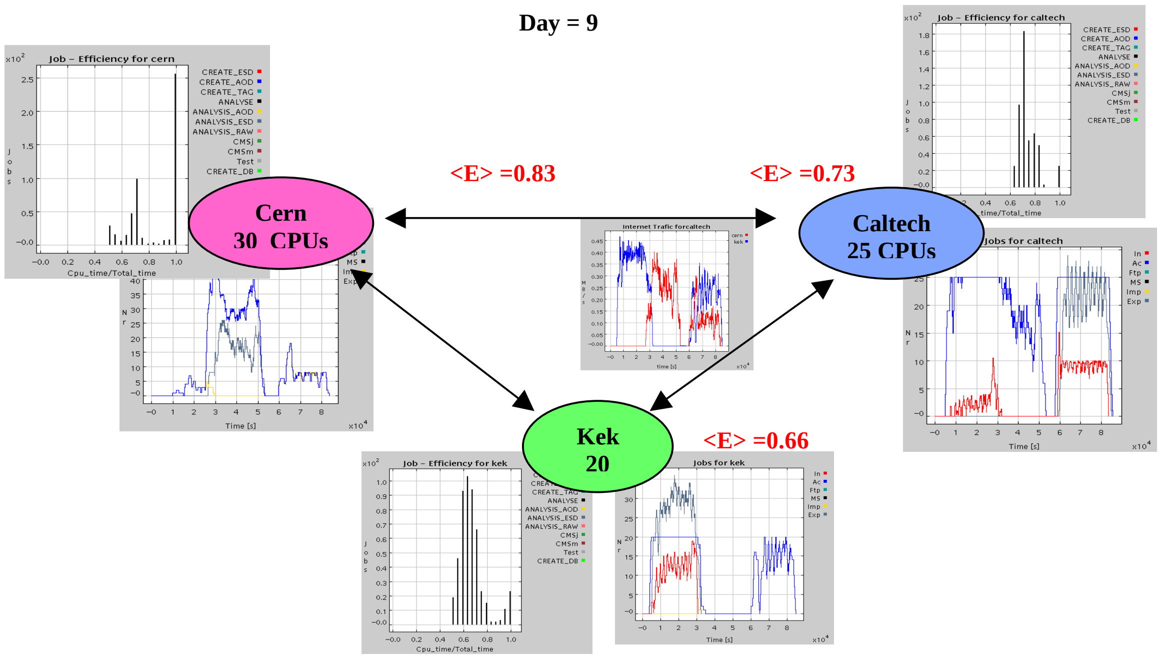

Figure 8. The job evolution (for one Day=9) and the distribution of the efficiency turnaround time after several days of training.

The improvement in the mean turnaround time for both regional centers during this learning procedure is presented in figure 9.

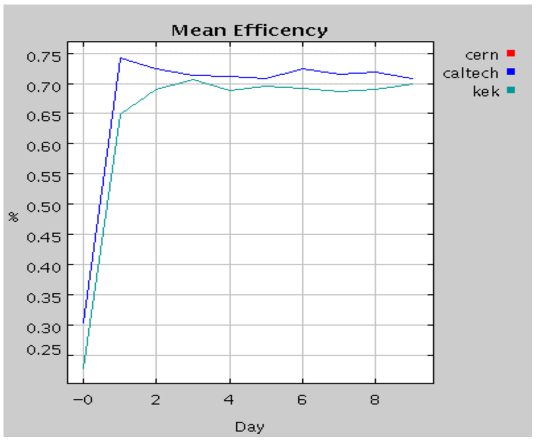

Figure 9. The improvement of the mean job efficiency (CPU time/ total time) or the turnaround time, during the learning process.

8. Summary

Using such a scheduling approach on quite simple problems in a realistic environment provided by the MONARC simulation tool, seems to offer a possible solution for distributing jobs between Regional Centers. As mentioned before, the difficult part in this approach is not the learning part but the decision making scheme which needs to find efficient ways to explore the “unknown” parts of the parameter space. It is also important to mention that, for any specific stochastic optimization problem, such an approach is not a guarantee to find the “best possible solution”.

References

[1] MONARC Simulation Tool http://www.cern.ch/MONARC/sim_tool/

[2] B. Fritzke, “A self-organizing network that can follow non-stationary distributions, “ Proc. of the International Conference on Artificial Neural Networks ‘97, Springer, 1997, pp. 613-618.
B. Fritzke. Growing self-organizing networks , In ESANN’96: European Symposium on Artificial Neural Networks, pages 61--72. D-Facto Publishers, 1996.

[3] M. Norgaard, O.Ravan, N,K, Poulsen and L.K. Hansen, Neural Networks for Modelling and Control of Dynamic Systems, Springer, 2000

[4] B. Fritzke. A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, Advances in Neural Information Processing Systems 7, pages 625--632. MIT Press, Cambridge MA, 1995.